

C Programming Notes

By: error_terminator

Content

Page No.

1. Array

2

2. Structure

11

3. Functions

17

4. Pointers

22

5. File Handling

25

Array

An array is a collection of similar Data type. It is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. All arrays consist of contiguous memory locations.

First Index

Last Index

Ar[0]	Ar[1]	Ar[2]	Ar[3]	Ar[4]	Ar[5]	Ar[6]
-------	-------	-------	-------	-------	-------	-------

Here are 7 elements in an Array Ar represents the name of Array.

The **First Index** and **Last Index** represent the index value starting from 0 to n-1, where n is the size of array.

Declaring Array

To declare an array we must specify the data type and the name of array, which will be required later.

datatype array_name[integer or constant value];

This is a *single-dimension array* in C. There are *multi-dimensional arrays* in C.

datatype must be any valid data type in C i.e. int, float, double etc. *array_name* should start with character. *integer or constant value* must be an integer greater than zero or a constant value that is declared earlier in your program.

int pages[5];

int is a valid data type in c , *pages* is the name of array, 5 is the integer value.

The size depends on your system it can be 2 or 4 bytes. Here the size for int pages is 5*4(in my system) = 20 bytes.

Initializing Array

You can initialize an array in one statement, can be written as –

```
int pages[5] = {1,2,3,4,5};
```

Array can be initialized in curly braces {}. If the number of elements in {} braces exceed the integer value it will throw an error. So the number of elements should be equal.

We can initialize with the less number of elements, but this will be a total wastage of memory.

```
int pages[ ] = {1,2,3,4,5,6};
```

As many as elements can be stored in the above expression, if no integer value is given.

0th Index

4th Index

1	2	3	4	5
---	---	---	---	---

The element at 0th index is 1 and at 4th index is 5.

We can take input from user in an array as:

```
#include <stdio.h>

void main()
{
    int array[5]; //declaring array with elements 5
    int i;
    printf( "\nEnter Array elements ");
    for(i=0; i<5; i++) //for loop for iteration
        scanf( "%d", &array[i]);
}
```

Accessing Array Elements

We can access array elements by using the index of each element or transverse through an array using for loop.

Let us consider the same array i.e. *int pages[5] = {1,2,3,4,5};*

Here *pages[0]* is 1, *pages[1]* is 2, *pages[2]* is 3 and so on...

Accessing elements through for loop is as follow :

```
int i;//for iteration
```

```
for(i=0;i<5;i++)//instead of 5 we can use n,but as we have array of 5 elements so we use 5
```

```
    printf("%d",pages[i]);
```

```
#include <stdio.h>
```

```
void main ()
```

```
{
```

```
    int arr[ 10 ]; // arr is an array of 10 integers
```

```
    int j;
```

```
    printf("Enter elements ");
```

```
    for (j = 0; j < 10; j++ )
```

```
        scanf("%d",&arr[j]);
```

```
    for (j = 0; j < 10; j++ ) { //printing elements
```

```
        printf("Element[%d] = %d\n", j, arr[j] );
```

```
    }
```

```
}
```

```
"D:\c programs\array.exe"
Enter elements 1
2
3
4
5
6
7
8
9
10
Element[0] = 1
Element[1] = 2
Element[2] = 3
Element[3] = 4
Element[4] = 5
Element[5] = 6
Element[6] = 7
Element[7] = 8
Element[8] = 9
Element[9] = 10

Process returned 0 (0x0)   execution time : 7.921 s
Press any key to continue.
```

Multidimensional Array

C allows user to use multidimensional array.

type name[size1][size2].....[sizeN];

Here type can be any valid data type in c and size 1 to size N be integer values. Also name be any valid c identifiers

For example;

`int array[5][5];`

This is a two dimensional array with name array and 5 rows and 5 columns.

Two dimensional arrays are the simplest form of multidimensional array in C

type array_name[i][j];

Where **type** can be any valid C data type and **array_name** will be a valid C identifier. A two-dimensional array can be considered as a table which will have x number of rows and y number of columns.

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Initializing Two-Dimensional Arrays

The multidimensional array can be initialized as

```
int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
```

Here 5,2 is the i^{th} row and j^{th} column(can be simply called for understanding).

Accessing Two-Dimensional Arrays

Two-Dimensional Array can be accessed by using row and column index.

```
#include <stdio.h>

int main () {
    /* an array with 5 rows and 2 columns*/
    int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
    int i, j;
    /* output each array element's value */
    for ( i = 0; i < 5; i++ ) {
        for ( j = 0; j < 2; j++ ) {
            printf("a[%d][%d] = %d\n", i, j, a[i][j] );
        }
    }
}
```

Passing Array to Function

Passing array to function is simple, Let us consider an array

```
int salary[10];
```

Let this array stores the salary of 10 peoples. And we want to pass it as an argument to a function, so we can do in the following way as shown below:

Function declaration

```
void new(int );//No return type and name of function is new taking parameter as int
```

Function call

```
new(salary);//Passing array, calling function by using its name and passing array
```

While **defining** we have to write syntax as shown below

```
void new(int salary[])//Accepting array
```

```
void new(int salary[5])//Also accept array in this type
```

Actually in c, the whole array is not passed but, only the address of first array element is passed as a parameter or argument. While accepting array as a parameter we can pass the pointer to the array.

```
new( *salary);
```

Note: Here new is the name of function with void return type.

```
#include <stdio.h>
```

```
void new(int arr[5], int n )
```

```
{
```

```
    int i;
```

```
    for(i=0;i<n;i++)
```

```
        printf("%d",arr[i]);
```

```
}  
  
void main()  
{  
  
    int arr[5]={1,2,3,4,5};  
  
    int n = 5;  
  
    new(arr,n);  
  
}
```

Output for the above program will be

12345

Pointer to Array

C allows the usage of pointers i.e. address of an element.

Assuming an array with name array and size 20 bytes storing 5 elements of int data type.

```
int array[5];
```

Assign a pointer to it (pointers in c)

int *ptr; //Here *ptr stores the address of the first element of array and the other elements can be accessed by *(ptr + 1),*(ptr + 2) and so on...

```
#include <stdio.h>
```

```
void main () {
```

```
/* an array with 5 elements */
```

```
int array[5] = {1, 2, 3, 4, 5};
```

```
int *p;
```

```
int i;
```

```
p = array; //also p = &array; & is dereferencing operator
```

```
/* output each array element's value */
```

```
printf( "Array values using pointer\n");
```

```
for ( i = 0; i < 5; i++ ) {
```

```
printf( "\n%d\n", *(p + i) );
```

```
}
```

```
printf( "Array values using array as address\n");
```

```
for ( i = 0; i < 5; i++ ) {
```

```
printf( "\n%d\n", *(array + i) );
```

```
}
```

```
}
```

Output:

1 2 3 4 5 and 1 2 3 4 5

In the above example `*p` is a pointer of type `int` and it will store address of first element in `p`. Once we stored the address its easy to access `*p`.

Note:

If an array is not initialized i.e. only declared, some compilers (depending on system) gives value as 0 and some garbage value.

error — terminator

Structure

As like array we have structure in C, Arrays are used to store elements of same data type, while structure allows us to store elements of different data type with a name assigned to a structure.

Suppose we want to store an information of book such as Title,pages,price we can't store it in array, So we have structure.

Defining structure

For declaring structure we have a keyword *struct*

Syntax:

```
struct [structure tag] {  
  
    member definition;  
    member definition;  
    ...  
    member definition;  
} [one or more structure variables];
```

Here structure tag is optional and struct is keyword we can have one or more structure variables, same as a class in Java or C++ .

For ex:

```
struct //keyword for struct  
  
{  
  
    int pages; //integer type pages  
  
    char name[20]; //name of book type string  
  
    float price; //price as float  
  
    }b1, b2; //two or more structure variables
```

Accessing Structure Variables

To access any member of a structure, we use the *member access operator* (.). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword *struct* to define variables of structure type.

Here's a program for better understanding

```
#include <stdio.h>

#include <string.h>

struct book //defining structure with structure tag book
{
    int pages; // int type pages
    float price; //float type prices
    char title[50]; // string title
    char name[50]; //string name
};

void main()
{
    struct book b1;

    b1.pages = 350; //initializing members of structure with . operator
    b1.price = 150.3;
    strcpy(b1.title, "Let Us C"); //strcpy from string.h
    strcpy(b1.name, "Yashwant Kanhetkar");

    printf("\nPages:%d\nPrice: %.2f\nTitle:%s\nName:%s\n", b1.pages, b1.price, b1.title, b1.name); //printing members
}
```

```
"D:\c programs\structure.exe"

Pages:350
Price:150.30
Title:Let Us C
Name:Yashwant Kanhetkar

Process returned 63 (0x3F)    execution time : 0.024 s
Press any key to continue.
```

Structure as Function Argument

You can pass structure to a function in the way we pass any variable.

```
#include <stdio.h>
#include <string.h>

struct book
{
    int pages;
    float price;
    char title[50];
    char name[50];
};

void print(struct book );//Function Declaration

void main()
{
    struct book b1;
```

```
b1.pages = 350;
b1.price = 150.3;
strcpy(b1.title, "Let Us C");
strcpy(b1.name, "Yashwant Kanhetkar");
print(b1); //function call
}

void print(struct book b1) /*function definition no need to use same name of variable b1 or
b2 we can assign any name*/
{

printf("\nPages:%d\nPrice: %.2f\nTitle:%s\nName:%s\n", b1.pages, b1.price, b1.title, b1.name);
}
```

The output for the above program will be:

```
"D:\c programs\structure.exe"

Pages:350
Price:150.30
Title:Let Us C
Name:Yashwant Kanhetkar

Process returned 63 (0x3F)   execution time : 0.024 s
Press any key to continue.
```

Pointers to Structure

You can define a pointer variable of struct as we do for other pointers.

```
struct book *ptr;
```

Now we can store an address into ptr by using & operator

```
ptr = &b1;
```

For accessing we use -> operator (of operator, a dash – and greater than sign >)

```
ptr->title;
```

Here we have to use different names for the variables as we are storing address of a variable into another pointer.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
struct book
```

```
{
```

```
    int pages;
```

```
    float price;
```

```
    char title[50];
```

```
    char name[50];
```

```
};
```

```
void print(struct book *b); //Function Declaration
```

```
void main()
```

```
{
```

```
    struct book b1;
```

```
    b1.pages = 350;
```

```
b1.price = 150.3;  
strcpy(b1.title, "Let Us C");  
strcpy(b1.name, "Yashwant Kanhetkar");  
print(&b1);//function call  
}  
void print(struct book *b)//function definition  
{  
    printf("\nPages:%d\nPrice:%.2f\nTitle:%s\nName:%s\n",b->pages,b-  
>price,b->title,b->name);  
}
```

Output will be the same;

Functions

A function is building block of any program, we at least have one function in our program i.e. `main()` . We use function for simplifying our code. You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function *declaration* tells the compiler about a function's name, return type, and parameters. A function *definition* provides the actual body of the function.

There are four types of functions:

1.with return_type with argument

Ex: `int print(int);`

2. with return_type no argument

Ex: `int print();`

3. no return_type with argument

Ex: `void print(int);`

4. no return_type no argument

Ex: `void print();`

Defining a Function

A function is declared with its return type, name of function and argument (also called as parameter).

Syntax:

```
return_type function_name(argument);{  
  
body;  
  
}
```

Function Declaration

You have to declare a function that you're using in your code, to tell compiler which function you are using. A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

Syntax:

return_type function_name(parameters);

We have two types for declaration;

- 1.return_type function_name (int num);//declaring int num*
- 2.return_type fun_name (int);//declaring name of the variable is not mandatory but data type is*

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

Calling a Function

While creating a function we specify, what task it has to do, so when calling it will perform the specific task defined into a function.

Syntax:

- 1.function_name(parameter_list);//NO RETURN_TYPE for function*
- 2.result = function_name(parameter_list);// RETURN_TYPE for function*

Here result is just a name of variable and can be used any, depending on your return_type if function.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

```

#include <stdio.h>

int max(int , int );//function declaration

int main () {

    int a = 10;

    int b = 20;

    int res;

    /* calling a function to get max value */

    res = max(a, b);//function call

    printf( "Max value is : %d\n", res );

    return 0;

}

/* function returning the max between two numbers */
int max(int num1, int num2) {

int result;//variable limited to function only i.e. local variables

if (num1 > num2)

    result = num1;

else

    result = num2;

return result; //returning result of type int

}

```

Output:

Max value is : 20

Function Arguments

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function there are two ways of passing argument;

- 1.Call by Value
- 2.Call by Reference

Call by Value

The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

By default, C programming uses *call by value* to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

Function definition to swap the values

```
void swap(int x, int y) {  
    int temp;  
    temp = x;  
    x = y;  
    y = temp;  
}
```

If we print the argument passed from main() into the swap() function there will be no change in the values of main(), but only change in values of function.

//try to run code with main()

Call by Reference

The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument.

To pass a value by reference, argument pointers are passed to the functions just like any other value. So accordingly you need to declare the function parameters as pointer types as in the following function swap (), which exchanges the values of the two integer variables pointed to, by their arguments.

```
void swap(int *x, int *y)
```

```
{//function definition for swapping values using address.
```

```
    int temp;
```

```
    temp = *x;
```

```
    *x = *y;
```

```
    *y = temp;
```

```
}
```

Here the address of variable is passed as an argument, this affects the original value.

While calling the function, we must call it with

```
swap(&a, &b); //& dereferencing operator
```

Pointers

Pointers in C are easy to learn. Tasks are more easily performed in pointers such as call by reference, dynamic memory allocation, etc. C allows the use address of the variable to be stored in a special variable named pointer. As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address.

```
#include <stdio.h>

int main ()
{
    int var;

    printf("Address of var variable: %x\n", &var );
}
```

The above code will print the address of var.

Output: Address of var variable: 60fefc (Memory Location will change with multiple tests)

Declaration of a Pointer

The general way to declare a pointer is:

```
data_type *var_name;
```

Here, data_type is the pointer's base type; it must be a valid C data type and var_name is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer.

Here are some of pointers declared

```
int    *ip;   /* pointer to an integer */  
double *dp;   /* pointer to a double */  
float  *fp;   /* pointer to a float */  
char   *ch    /* pointer to a character */
```

Using Pointers in programs:

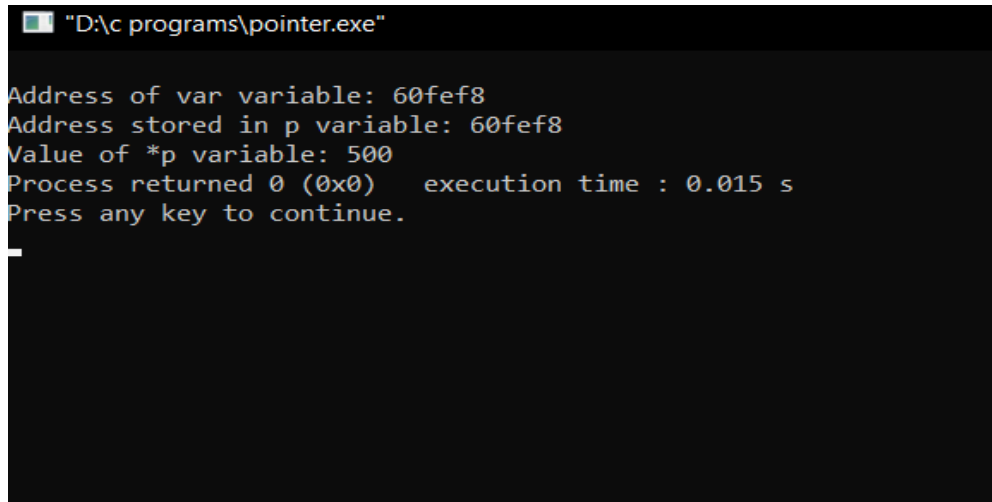
Using pointers is easy in C, Steps to use pointers in your program:

- (1) We define a pointer variable
- (2) Assign the address of a variable to a pointer
- (3) And finally access the value at the address available in the pointer variable.

Program:

```
#include <stdio.h>  
  
void main ()  
{  
    int var = 500; /* variable declaration */  
    int *p;        /* pointer variable declaration */  
    p = &var; /* store address of var in pointer variable*/  
    printf("\nAddress of var variable: %x", &var );  
    /* address stored in pointer variable */  
    printf("\nAddress stored in p variable: %x", p );  
    /* access the value using the pointer */  
    printf("\nValue of *p variable: %d", *p );  
}
```

Output for the above program:



```
"D:\c programs\pointer.exe"
Address of var variable: 60fef8
Address stored in p variable: 60fef8
Value of *p variable: 500
Process returned 0 (0x0)   execution time : 0.015 s
Press any key to continue.
```

NULL Pointers

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a *null* pointer.

The value of ptr is 0 only if it's set to NULL.
i.e. ptr = NULL;

File handling

A file is a container in computer storage devices used for storing data.

Types of Files

When dealing with files, there are two types of files you should know about:

Text files

Binary files

Different operations that can be performed on a file are:

1. Creation of a new file (**fopen with attributes as “a” or “a+” or “w” or “w+”**)
2. Opening an existing file (**fopen**)
3. Reading from file (**fscanf or fgetc**)
4. Writing to a file (**fprintf or fputs**)
5. Moving to a specific location in a file (**fseek, rewind**)
6. Closing a file (**fclose**)

Opening a file - for creation and edit

Opening a file is performed using the `fopen()` function defined in the `stdio.h` header file.

The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen", "mode");  
fopen("E:\\cprogram\\newprogram.txt", "w");
```

Closing a File

The file (both text and binary) should be closed after reading/writing.

Closing a file is performed using the `fclose()` function.

```
fclose(fptr);
```

Here, `fptr` is a file pointer associated with the file to be closed.

Ex 1. Writing to a .txt file

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;
    FILE *fptr;

    // use appropriate location if you are using MacOS or Linux
    fptr = fopen("C:\\program.txt","w");

    if(fptr == NULL)
    {
        printf("Error!");
        exit(1);
    }

    printf("Enter num: ");
    scanf("%d",&num);

    fprintf(fptr,"%d",num);
    fclose(fptr);

    return 0;
}
```

This program takes a number from the user and stores in the file `program.txt`. After you compile and run this program, you can see a text file `program.txt` created in C drive of your computer. When you open the file, you can see the integer you entered.

Ex 2. Reading to a txt file

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int num;
    FILE *fptr;

    if ((fptr = fopen("C:\\program.txt", "r")) == NULL){
        printf("Error! opening file");

        // Program exits if the file pointer returns NULL.
        exit(1);
    }

    fscanf(fptr, "%d", &num);

    printf("Value of n=%d", num);
    fclose(fptr);

    return 0;
}
```

This program reads the integer present in the `program.txt` file and prints it onto the screen.

If you successfully created the file from **Example 1**, running this program will get you the integer you entered.

Other functions like `fgetchar()`, `fputc()` etc. can be used in a similar way.

References:

Variety of content from different web sites were used for making this notes the best one.

1.Programming with C – R.S.Bichkar

2.TutorialsPoint

3.GeeksForGeeks

4.ProgrammingBiz

error – terminator